Subject: Re: [PATCH] usbatm: Update to use the kthread api.
Posted by Duncan Sands on Thu, 14 Dec 2006 13:14:38 GMT
View Forum Message <> Reply to Message

Hi Eric,

> > ...
> > Once you accept that a signal needs to be sent, you can't remove all
> > those completions etc that your patch deleted, because it introduces
> > races: they are there to make sure that (a) signals are unblocked in the
> > thread before the signal can possibly be sent, and (b) the signal is not
> > sent to the wrong thread if the kernel thread exits at a badly chosen
> > moment and the pid is recycled. I believe the current setup is race
> > free, but please don't hesitate to correct me. If you want to get rid
> > of the pid and instead use a pointer to the thread then avoiding race (b)
> > becomes even more important, since then rather than shooting down the wrong
> > thread you send a signal to a thread which no longer exists, surely a fatal
> > mistake.
>
> Actually I don't accept that a signal needs to be sent. I do accept
> that the message needs to be delivered to stop things early.

I'm not in love with signals either, however...

> The paradigm in a kthread world for waking up kernel threads is by
> calling kthread_stop, and then for testing if a kernel thread should
> stop is by calling kthread_should_stop.

I considered this, but rejected it because of this comment:

```
 * kthread_stop - stop a thread created by kthread_create().
 * ... Your threadfn() must not call do_exit()
 * itself if you use this function! ...
```

and this one:

```
 * ... @threadfn can either call do_exit() directly if it is a
 * standalone thread for which noone will call kthread_stop(), or
 * return when 'kthread_should_stop()' is true (which means
 * kthread_stop() has been called).
```

Most of the time the kernel thread starts, performs heavy_init,
and exits. The above comments seem to imply that it is wrong
to call do_exit if kthread_stop might be called, and wrong to
return if kthread_stop has not been called. This seems to exclude
the case where kthread_stop is sometimes, but not always, called,
and the thread sometimes exits without kthread_stop having been
called. But perhaps I misunderstood, since it seems there is kthread

code to handle the case of a threadfn that returns without kthread_stop
having been called, witness this comment:
        /* It might have exited on its own, w/o kthread_stop. Check. */
It's still not clear to me, so if you can enlighten me, please do!

> Especially if you are looking at generalizing this code over all of
> usb it should probably be using the current kernel best practices.
>
> There is still an issue with msleep here that I completely concede.
> In particular neither msleep nor msleep interruptible will actually be
> awoken by kthread_stop. So it looks like we need a msleep_kthread
> that will won't go back to sleep if after kthread_stop wakes it up.
> Still unless I am blind that looks like a very minor change from where
> we are now.

Sure.

> I think the reduction in complexity and the increase in uniformity
> is most likely worth it.
>
> If all else fails I'm happy with something simpler like Cedric's
> patch which takes care of the things that I currently have a problem
> with, but I'm willing to work through this to make it a through
> cleanup.

You have a problem with the pid, right? Well, that is easily
cured in itself. I'll spin a patch for it a bit later, unless
someone else gets there first. And if you can confirm that kthread_stop
can be used in this situation (i.e. thread can spontaneously return
without kthread_stop) then I'm happy to convert everyone over to checking
kthread_should_stop.

Ciao,

Duncan.